Naval Research Laboratory

Washington, DC 20375-5320

NRL/MR/8140--97-7945

Network and System Management of ICEbox through SNMP

DANIEL C. LEE

Command Control Communication Computers and Intelligence Branch
Space Systems and Development Department

M. EDWIN JOHNSON

Contractor for Code 8140

April 30, 1997

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (*Leave Blank*)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	April 30, 1997	

4. TITLE AND SUBTITLE

Network and System Management of ICEbox through SNMP

5. FUNDING NUMBERS

6. AUTHOR(S)

Daniel C. Lee and M. Edwin Johnson*

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Naval Research Laboratory
Washington, DC 20375-5320

8. PERFORMING ORGANIZATION REPORT NUMBER

NRL/MR/8140--97-7945

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

SAF/FMBMB (AFOY)
Washington, DC 20050-6335

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

*Contractor for Code 8140

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT (*Maximum 200 words*)

This report discusses the design and implementation of an experimental network management system. This system can monitor both the networks that employ the standard SNMP and the ICEbox system that employs a private management protocol. ICEbox (Improved Collection Equipment) is a system developed by the Naval Research Laboratory for data collection, processing, and dissemination. This report describes the private management protocol of ICEbox and its software implementation. Then, this report presents the design and implementation of the experimental network management system's software modules and the proxy agent that provide the interface between ICEbox and the SNMP-based network management software. In designing the proxy agent, a commercial development tool, EventIX was used. In the course of discussion, this report illustrates the use of this tool and also evaluates the tool. The work is motivated by the need to monitor networks employing different management protocols from a consolidated station. In the consolidated management system described in this report, the status of many independently developed systems are presented in a single display station for monitor control. This report also shares experiences of using the commercial management software Open View Network Node Manager in the consolidated management station.

14. SUBJECT TERMS

Proxy agent
Interoperability
Consolidated management

15. NUMBER OF PAGES

22

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

CONTENTS

NETWORK AND SYSTEM MANAGEMENT OF ICEBOX THROUGH SNMP

I. INTRODUCTION

This report stems from experiences in designing and implementing a particular network management system. In this management system, the status of many independently developed systems are combined into a consolidated display for monitor and control. The commercial off-the-shelf (COTS) network management software, HP OpenView Network Node Manager[1] was employed for this consolidated display. Among the independently developed systems, we focus on the ICEbox system. ICEbox is a large-scale system for data collection, processing, and dissemination developed by the Naval Research Laboratory. While many commercial network management software products, including Network Node Manager, use the Simple Network Management Protocol (SNMP) [Ros94] to exchange management information, ICEbox does not communicate network and system management information using SNMP. Therefore, proxy agent technology [Ros94, Fei95] was adopted to interface ICEbox to the consolidated management system. For the design and implementation of the proxy software, a COTS development tool, EventIX[2], was used.

This report will present the design of the proxy agent for accessing ICEbox status data and evaluate the design. The evaluation will focus on this design's ability to allow "high-level" ICEbox status information to be available via SNMP. ("High-level status information" here means information other than standard network management information defined in the Management Information Base (MIB). The examples of such high-level information would be application status, process status, mission status, system status, etc.) This report will also discuss experiences in using the two commercial products for the design and implementation. This report will introduce and comment on these tools. Regarding the OpenView Network Node Manager, pros and cons of using OpenView to display ICEbox high-level status information will be discussed.

II. ARCHITECTURE OF CONSOLIDATED MANAGEMENT

We envision a central management station that can monitor and control several networks including the ICEbox network (Figure 1) The central station uses the Simple Network management Protocol (SNMP) for the exchange of the management information. Because the ICEbox does not use SNMP, a proxy agent [Ros94,Fei95] needs to be designed to translate between the manager and ICEbox. In designing this architecture, we decided to use the COTS products and development tools as much as possible. We used the Hewlett Packard's OpenView Network Node Manager as the network management software. For development of proxy software we used Bridgeway Corporation's EventIX. EventIX is a software that aims at developing a consolidated network management system for an environment wherein different legacy management systems are in use. The goal of our project is basically to consolidate the ICEbox's management system and the standard management system based on SNMP, so EventIX was deemed a proper tool of choice.

---

[1]OpenView Network Node Manager is a registered trademark of Hewlett-Packard Company.
[2]EventIX is a trademark of Bridgeway Corporation.
Manuscript approved March 18, 1997.

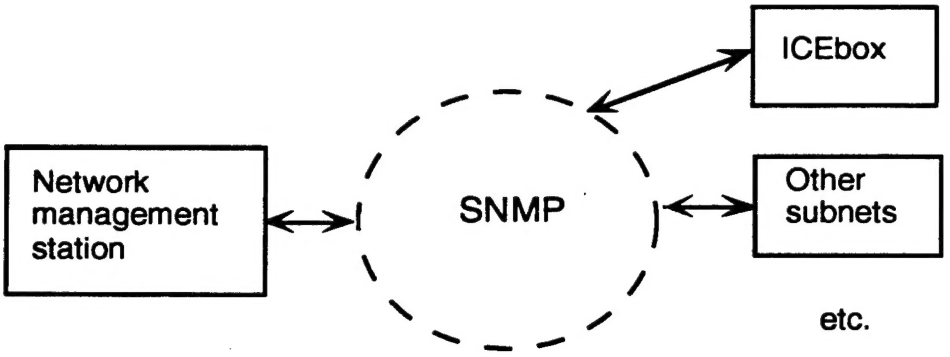


Figure 1: Consolidated Network Management

III. ICEBOX

In order to meaningfully present the SNMP proxy agent design for ICEbox, some knowledge of ICEbox system must be conveyed. ICEbox (Improved Collection Equipment) is a system of geographically separated mobile hardware and software that is used to collect, communicate, and process data. ICEbox can be viewed as a communication network and/or distributed processing system. This report focuses on the software modules for communication, control and management of ICEbox system.

A. Modules

ICEbox design is modular in nature. An ICEbox module is a collection of software tasks and/or hardware that hides and abstracts an aspect (typically a specific function) of the ICEbox system from the rest of the system. Modules are recognized at a system level; that is, they are the first layer of decomposition of the ICEbox system. Due to the modular design, ICEbox capability can be extended by adding new modules to the existing ICEbox system. Modules must communicate with each other. Communication infrastructure connecting the modules is referred to as ICEnet. Each module has a unique module ID. When a programmer makes a new module to expand ICEbox capability, the new module must be reported to the ICEbox administrator, and the administrator assigns a module ID. ICEbox may have multiple instants of a particular module. Each instant must be uniquely addressable via ICEnet. Therefore, each instant is assigned a system ID, and together the system ID and the module ID form a unique ICEnet address. The source and destination in the inter-module communication is specified by this ICEnet address.

B. ICEbox Control and Management System

The heart of the control system is the module referred to as Element Control (EC) module. Element Control module can be programmed by a human operator, and the entire ICEbox system can be controlled by the EC module without human presence. In order to control and coordinate the data collection activities, the EC gives proper commands to other modules and gathers, collates, processes the reports from the other modules. In particular, EC module frequently (every one or five seconds) receives the status report from other modules and keeps the status information. Another important module for ICEbox management is the

Human Computer Interface (HCI). HCI can be viewed as the network management software for the ICEbox. HCI periodically (every three seconds) polls the EC to obtain updated status information for all modules and graphically displays the status of ICEbox modules to the host's monitor so that an operator can observe ICEbox status. From HCI, an operator can control ICEbox and schedule the ICEbox's data collection activities. network. The EC and HCI exchange status and control messages thorough ICEnet. (Figure 2)

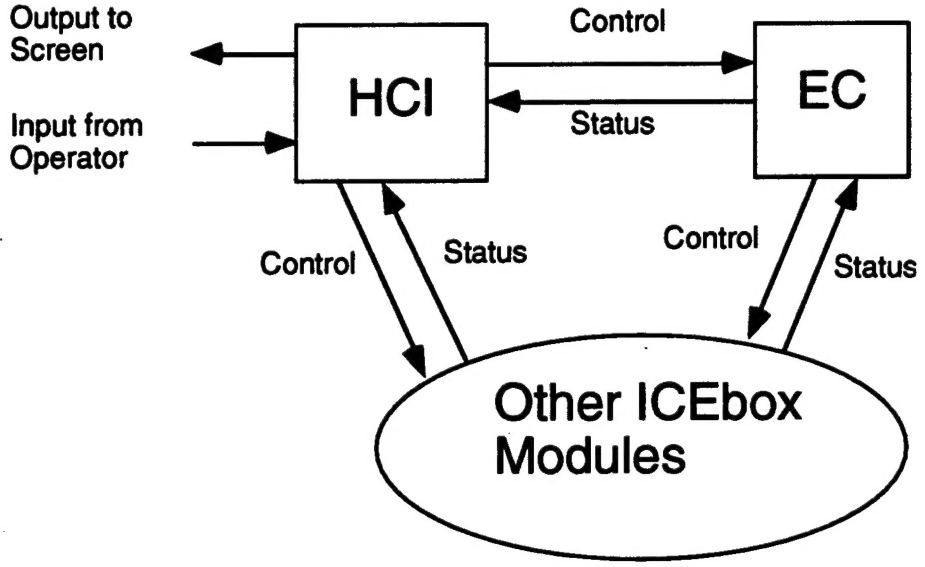


Figure 2: ICEbox Control System

For control and management, ICEbox takes the same approach as the Simple Network Management Protocol (SNMP); fetch--and--store paradigm. [Com91] Each ICEbox module contains a number of data items which can be modified or examined by the ICEbox control system. Such a data item in ICEbox is referred to as "Control Object Documented" (COD). CODs are like MIB (Management Information Base) [Ros94] variables of the SNMP paradigm. EC module monitors the status of another module by examining the contents of the monitored module's CODs. (Fetch) EC module controls another module by changing the contents of the controlled module's CODs. (Store) The contents of these Control Objects (CODs) are conveyed between modules in a message format referred to as the "Inter-device Command Block Message" (ICBM). (The communication among modules will be explained more in detail in a later section.) In the ICBM format, there is a number of fields including source/destination address and message type. The communicating modules write message types such as Read, Read Ack, Write, Write Ack in the message type field when they send the ICBM message to the destination module. In order to monitor a certain aspects of a module, say module X, EC module sets the message type field with "Read" and specifies which COD it wants to read in the payload of ICBM. Then, it sends the ICBM message to Module X. Then module X replies to the EC with Read Ack type of message that provides the information of the requested COD. Similarly, in order to control a module, the EC sends Write message with certain control information. Then the module stores that control information in the proper COD and replies back to the EC with Write Ack.

C. Software Environment and Inter-module Communication

We have briefly discussed conceptual aspects of ICEbox modules and their management. In this section we discuss the software implementation more in detail.

C.1 Tasks, Services, Common Software

In order to have a clear discussion, we need to define certain terminology. A software "task" is a computer memory resident image of executable instructions for a host processor. ICEbox tasks typically are created from C source code procedures stored in multiple files that are compiled and linked to form an executable image. A task that performs a specific function for a module rather than a generic function for all modules is called an application task. The person who creates an application task is called the application programmer. A module runs one or more tasks during the execution. The ICEbox system has software that implements the inter-module interface mechanism. This is referred to "Common Software". The ICEbox Common Software provides a software development/execution environment for ICEbox modules. This environment for the most part is platform independent. This environment provides the application programmer with an abstract view of the services of the host operating system and network.

A service is a documented interface that performs a function for a task. Such functions include printing and sending and receiving messages. ICEbox Common Software provides those required services common to all tasks. These services are available to programmers in two ways; compile-time library of services and run-time tasks. For a programmer to request certain services to the common software, a module can call functions such as co_init_module, os_init, ib_get_icbm, ib_send_icbm, os_read_q, ib_rcv_icbm, etc. These functions are available as a library. Also, there are the run-time tasks that support all modules. These are called Common Software Tasks. Such tasks include "CEAR", "SEND", RECV", as will be explained in later sections. When a programmer executes an ICEbox module in a host system, those common software tasks must be running.

The design of ICEbox Common Software puts certain constraints on the choice of host system and the application programming. The host operating system (OS) must be "multi-tasking". The host operating system should support sending task-to-task messages via asynchronous message queues (ASQ). The ASQ is the backbone of the task-to-task communication of the Common Software. In order for a task to send a message to a second task's ASQ, the identification (id) number of the ASQ must be known to the sending task. All Common Tasks are to be considered "well known" in that their names and functions are assumed to be known by the application programmer. An application task must first provide its ASQ id to common tasks in order to exchange messages with them.

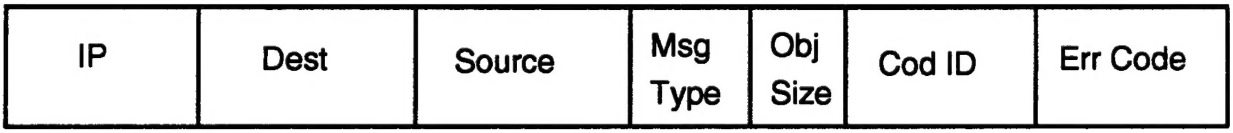
The host operating system should allow the utilization of "named semaphores" and "named shared memory segments" for task-to-task communication.

C.2 Inter-module Communications

The common software provides delivery and mail services of messages between ICEbox modules. Physical medium may be Ethernet bus, IEEE-488, RS232, or simply inter-task within a host. The Common Software hides the actual delivery mechanism from the modules by providing services to send and receive messages. The medium of all messages regardless of their actual path is considered to be ICEnet. ICEnet is a conceptual or virtual network connecting ICEbox modules.

When a new module is created and added to the ICEbox system, the module must make ICEnet aware of its location before it can receive messages from other modules. Common Software provides a service to allow a module to announce its presence. After announcing its existence, the module may now send and receive messages. All inter-module messages contain source and destination fields. To respond to a message, a module simply swaps source and destination and calls the send service. For communication media like Ethernet, ICEbox utilizes the internet protocols TCP/IP to send messages. The Common Software implementation utilizes an abstraction of the TCP/IP protocol called sockets. All target platforms for ICEbox are required to provide the socket interfaces defined in 4.3 BSD UNIX. The socket interfaces provide a mechanism for transferring IP datagrams (UDP). The Common Software adds a layer of abstraction to the IP datagram. This layer is called the *ICBM.* ICBMs contain the data required to transport itself. This data, being device dependent, is different for the Ethernet, IEEE-488, RS-232, etc. In the case of Ethernet, this data is an IP address.

An ICBM is the unit of data transfer between modules. These modules can be located anywhere on ICEnet. The ICBM structure contains the header (ICBM_HEADER) followed by the data. Figure 3 shows the structure of the ICBM_HEADER. (In fact, there are more fields in the header. Figure 3 is a simplified description.) ICBM is like the SNMP message.



ICBM_HEADER Structure

Term	Meaning
IP	The IP address of the source. 4 bytes.
DEST	The destination module's ICEnet address. 2 bytes.
SOURCE	The source module's ICEnet address. 2 bytes.
MSG_TYPE	Type of ICBM. 1 byte. Type could be Read, Read Ack, Write, Write Ack, etc.
OBJ_SIZE	The number of data bytes to follow in the payload. 2 bytes.
COD_ID	Identification of COD. 2 bytes. Refer to the Standard COD.
ERR_CODE	ICBM specific error codes. 4 bytes.

Figure 3. ICBM_HEADER Structure.

ICBM has the fields for Destination address, source address, message types, message, error codes, etc. There are several message types, e.g. Read, Read Ack, Write, Write Ack. A module sends an ICBM to Common Software, and Common Software is then responsible for delivery of the ICBM as shown in Figure 4. Figures 5 and 6 show the parts inside the Common Software blob of Figure 4 that are involved in transferring an ICBM from one module across the IP to another module. As indicated, Common Software relies heavily on the services of the host system to transfer the message. Figure 6 shows the path an ICBM takes when both the source and the destination modules reside in the same host. Tasks "CEAR", "RECV", "SEND" are run-time Common Software. The task CEAR knows the destination tasks resides in the same host and routes it directly. For mediums other than IP (such as IEEE-488, serial ports, etc.), Figure 5 is still valid, except that the tasks SEND and RECV are replaced by tasks for that medium.

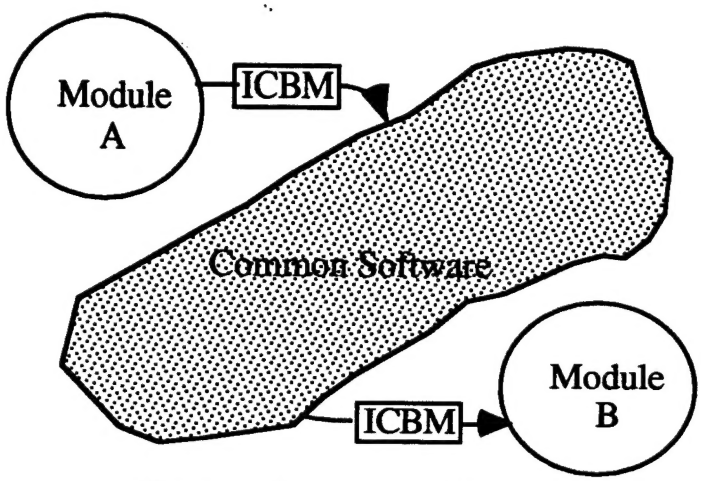


Figure 4: Sending an ICBM.

Task CEAR

Task CEAR has its own ASQ (asynchronous message queue), and it receives messages through the ASQ either from an application task or from Common Software task "RECV". To understand CEAR, one must first understand its routing scheme. Task CEAR utilizes two tables, IP_MAP and IP_OR_QID. These tables contain entries for every possible ICEbox module address. A module address consists of 7 bits to identify the module and 5 bits to identify the system id for a total of 12 bits or 4096 unique addresses. Table IP_MAP contains either the internet address or ASQ id of a module. Table IP_OR_QID is a logical table, where True means the messages to this module address go to the corresponding ASQ id entry in IP_MAP. False means that the messages to this address go to the internet address entry in IP_MAP. When CEAR initializes, it sets every entry of IP_OR_QID to False, and IP_MAP to the broadcast Internet address. Task CEAR receives messages from task RECV, modules, or possibly other device hiding tasks. When a message is received from task RECV, CEAR looks up the destination address in IP_OR_QID. If this table indicates an ASQ id, the message is routed to the ASQ id in IP_MAP. If the table indicates an Internet destination, the message is not routed anywhere. Every message received from task RECV includes the actual Internet address of the source. This address is written in table IP_MAP for that source so that

any writes to that source from this host are optimized. When task CEAR receives a message from any task other than RECV, the destination is looked up in IP_OR_QID. Those messages to go to ASQs are sent to the ASQ id of IP_MAP, and those messages to go to Internet addresses are sent to task SEND along with the Internet address from table IP_MAP.

In order for a module to receive messages from CEAR they must first announce their presence to CEAR. They do this by a supplied Common Software Service, ib_add_module, which is provided to the application programmer as a library function. The module calls this procedure passing the name of the module and the id of the ASQ to receive messages. When CEAR is passed with a module id and the module's ASQ id, CEAR updates table IP_OR_QID's entry corresponding to the module id to "True" and writes the ASQ id in table IP_MAP's line corresponding to the module id. If a new module announces its module id and ASQ id to CEAR, CEAR adds an entry in tables IP_OR_QID and IP_MAP. When a module's execution is terminated, the ib_del_module is called to remove the ASQ id from CEAR's IP_MAP table.

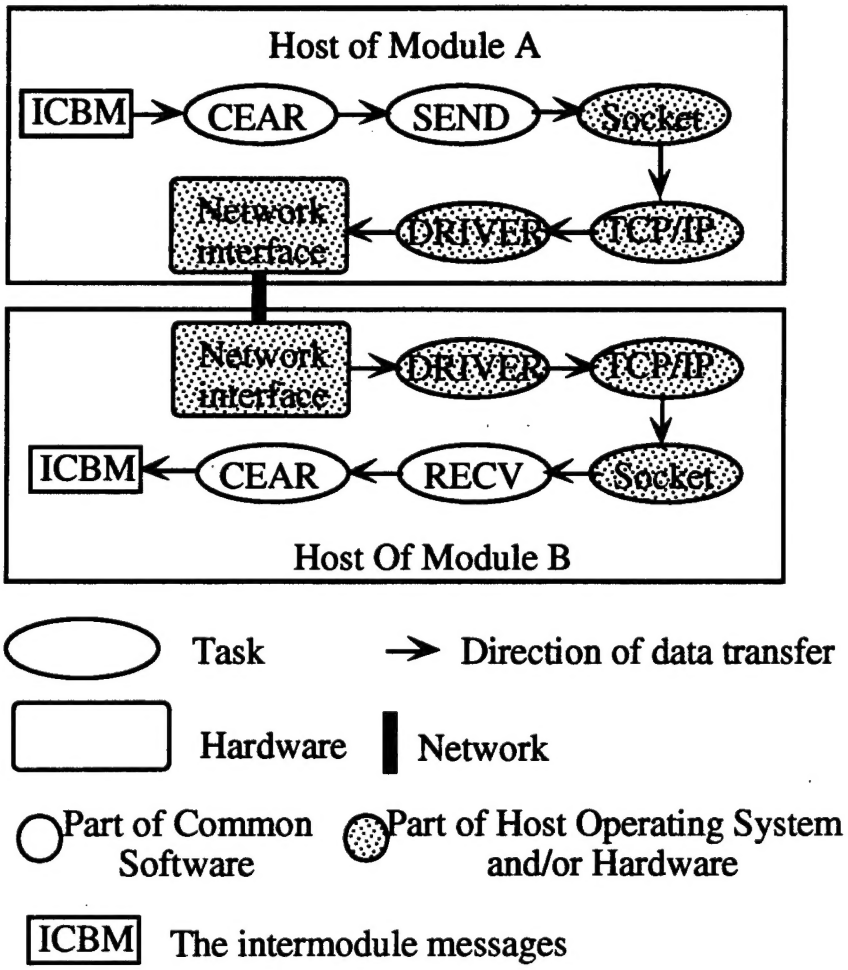


Figure 5. Common Software Sending an ICBM.

Figure 6. Sending an ICBM Intra-Host.

The routing scheme used by CEAR was chosen to allow the Common Software to learn the locations of all the modules. When a module sends a message to another module whose host location is not known, the message is broadcast on the Ethernet. The CEAR task on the host that contains the module will forward the message, all other hosts will ignore it. An ICEbox requirement is that a reply must be issued back to the sender of a message. When this happens, the CEAR task of the originating module will know the Internet address of the other module.

Task RECV

The RECV task waits endlessly for a message to be received on its socket. Upon receipt of the message, RECV checks the size of the incoming message by looking at the object size in the header. The message then is forwarded to task CEAR.

Task SEND

Task SEND waits endlessly for a message to arrive in its ASQ. Upon receipt of an message, SEND receives the ICBM. The ICBM is then sent out through a socket. Task SEND then waits for message at it's ASQ. A module places its ICBM in the ASQ of "CEAR", using a Common Software library function Also, a module reads its ASQ (by using again library routines of the Common Software) in order to receive the ICBM message from "CEAR".

IV. DESIGN AND IMPLEMENTATION OF SNMP-ICEBOX INTERFACE

In this section we describe the design and implementation of the interoperability between ICEbox system and SNMP-based management station. The design basically consists of an ICEbox module for this interoperability and an SNMP proxy agent for ICEbox system. We describe the design of the proxy agent and the design of how the proxy agent communicates with ICEbox.

A. Design

The role of the SNMP proxy agent for ICEbox is to forward the status information on ICEbox modules to the management station in response to the SNMP request from the station. To perform this role, the proxy agent must translate between ICEbox COD and SNMP MIB. The developer of the proxy agent must define MIB variables that correspond to ICEbox. (These MIB variables will not be standard but private. In our design we used iso.org.dod.internet.private branch of the MIB tree.) The developer must design the mapping between ICEbox COD information and SNMP MIB. The network manager in the central station must have the knowledge of which MIB variables describe which status of each ICEbox module in order to monitor and control ICEbox. The central management station may not be interested in microscopic details of the ICEbox status. The developer must identify which ICEbox status information must be available to the central management station and define MIB variables for that portion of ICEbox CODs. The design presented in this report allows the developers the flexibility to expand the scope of ICEbox information available to the central management station.

In ICEbox, the EC (Element Control) module periodically collects and maintains the status information of all modules. EC has a COD that contains such exhaustive ICEbox status information. The integration of ICEbox with other networks is desired to be as non-intrusive to ICEbox system as possible. Therefore, the proxy agent is designed to communicate only with EC module for monitoring the status of any ICEbox module, and not with any other ICEbox module. The proxy agent periodically gathers ICEbox status information from the EC module. Figure 7 illustrates the design. Poller in Figure 7 is a software module that periodically prompts a query message to be sent to an ICEbox module IEI. Module IEI (ICEbox EventIX Interface) is not a part of the established ICEbox system. It is an additional ICEbox module that is designed to interface the ICEbox with the proxy agent. While the commercial-off-the-shelf development kit greatly simplifies the implementation of the proxy agent, it cannot produce an ICEbox software module. EC can only communicate with another ICEbox module with an ICEnet address; that is, through ICBM and ICEbox Common Software. Therefore, we need to develop an ICEbox module that can communicate with the proxy agent also. IEI is the ICEbox module designed to interface between the proxy agent and the EC (Element Control). IEI sends Read-type ICBMs to EC upon receiving the query message from Poller. EC then replies to IEI with Read-Ack-type ICBM that contains status information of all ICEbox modules. IEI then forwards the contents of the status information to the proxy agent. The proxy agent then updates the MIB variables describing the status of ICEbox. The additional loading on ICEbox is minimal. The EC maintains status tables independent of data requests. Therefore, the only impact is a slight increase in network traffic between IEI and the proxy agent and between IEI and EC.

B. IEI (ICEbox EventIX Interface) module

IEI utilizes Common Software's library functions such as os_init and co_init_module for initialization service; thus, creating an asynchronous message queue (ASQ) for IEI, announcing the existence of IEI module to the Common Software. After initialization IEI opens a socket for receiving the polling message "do_query" from Poller and open another socket for sending messages to ice.epa. Then, IEI endlessly listens to the socket connection to Poller. If there is no message at the socket the process is blocked. Whenever it receives the polling message, IEI does:

- IEI makes up ICBM by filling up header fields; particularly, IEI writes two-byte address for EC module in the "Destination ICEnet address" field of the ICBM structure and "Read" in the "Message type" field. IEI sends this ICBM to the EC module. (According to the ICEbox' operational protocol EC replies by sending READ-ACK-type ICBM which contains the requested COD values.)
- IEI reads from its ASQ the ICBM replied from the EC.
- IEI then extracts the status information from the returned ICBM, makes up a string of text containing the status information of ICEbox , and sends the string to "ice.epa" module of the proxy agent as a datagram through the socket. IEI is written in C-language like other ICEbox software modules. (For sending/receiving ICBM and extracting information from ICBM, developers of IEI can use the ICEbox common software service. For example, library function ib_send_icbm sends an ICBM according to the mechanism described in section II.C.2.) In preparing the text string for the proxy agent (ice.epa module), IEI and the proxy

agent must agree in formats and which field of the string corresponds to which status information on which module of ICEbox. Thus, the programming for the proxy agent (ice.epa) and development of IEI must be done coherently.

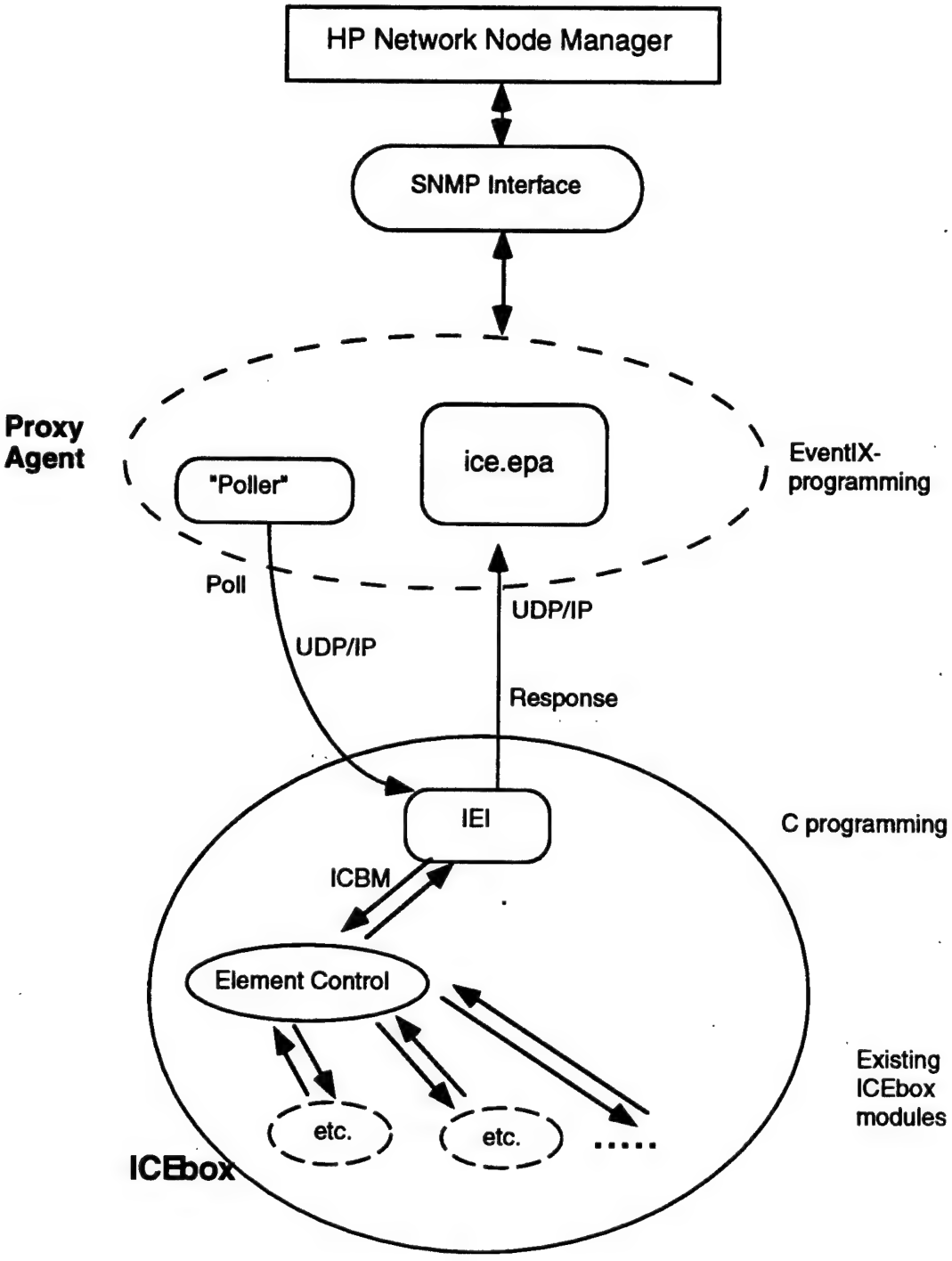


Figure 7: SNMP ICEbox Interface

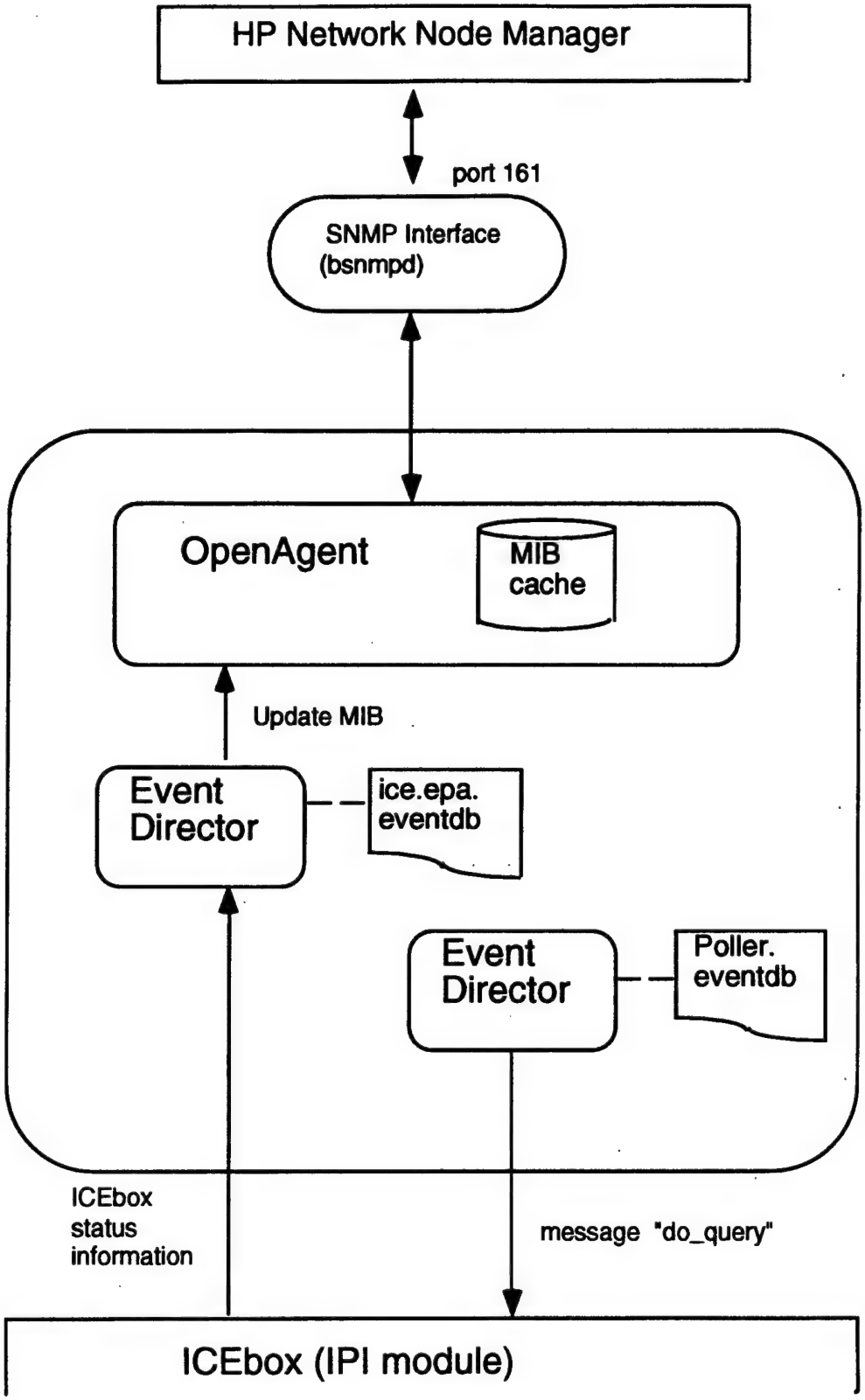


Figure 8: Proxy agent for ICEbox

C. Proxy agent

The proxy agent in our design basically :

- Receives from IEI the message containing the updated COD information.
- Updates MIB cache
- Processes query command from SNMP manager

In the previous sections and in Figure 7, the proxy agent was represented by two modules; Poller and ice.epa. In this section more details are presented. Figure 8 illustrates the architecture of the proxy agent. The proxy agent is composed of two run-time processes (OpenAgent and Event Director) and the software files (Poller and ice.epa) developed by the programmer. A commercial off-the-shelf tool, EventIX, was used for proxy agent development. Because EventIX was used as a tool, a brief introduction of EventIX is necessary.

EventIX is a collection of software that facilitates the exchange of data between different network management systems and simplifies the process of developing intelligent agents for network management. [Bri95] EventIX software products can be classified into two groups; development tools and run-time environment. Development tools are software-based facilities that allow users to build custom network management applications such as the proxy agent in a UNIX environment without low-level programming. It is important to note that EventIX employs event-driven programming paradigm. Programmers build an application software by defining events, actions, and rules that specify which actions must be taken in response to which event. During the run-time the software task basically takes certain "actions" in response to "events". An event is often the reception of certain messages by a UNIX process from another process. Actions could be starting another process, sending a message to another process, etc. Development tools provide a graphical user interface so that the programmer can conveniently specify the events, actions , and rules. The user-defined objects (events, actions, rules) are then stored in a file referred to as "eventdb" (meaning event database) file. Normally, a programmer assigns the file name followed by ".eventdb" in UNIX; e.g., poller.eventdb, ice.epa.eventdb for our proxy agent.)

Run-time environment incudes Event Director, OpenAgent, and the SNMP interface module. Event Director is an EventIX's built-in software program that relates all objects defined by the user and implements commands based on the actions defined. During the run time, Event Director is a UNIX process that detects and processes the user-specified events and takes actions according to the user-specified eventdb files. Event Director looks up the eventdb file and executes the prescribed actions in response to the defined events according to the rules.

SNMP Interface module is a run-time EventIX utility, which:

- Parses SNMP Get and GetNext messages coming from the network management station; forwards the parsed information to OpenAgent
- Constructs in SNMP format the messages based on the information received from OpenAgent and sends them to the management station

During the run time the SNMP Interface module is a UNIX process named as "bsnmpd" (daemon process). The bsnmpd listens to port number 161 as does the standard SNMP agent for SNMP Get Request or SNMP Set Request message.

OpenAgent is a run-time EventIX utility which retrieves information from the MIB Database cache and returns the response to the manager through bsnmpd. The following specific services are provided by OpenAgent:

- Caching MIB Database for SNMP information
- Receiving and processing updates to MIB Database cache.
- Retrieving information from MIB cache and communicate it to the SNMP Interface

Therefore, SNMP Interface and OpenAgent handles the SNMP messages from the Management station.

Both Poller and ice.epa in Figure 7 are written by using the development tools, and saved in eventdb files illustrated in Figure 8. Poller is a very simple application program, and the details of this program is presented to illustrate the event-driven programming used for developing the proxy agent. The events, actions, rules of Poller are as the following.

```
INITIALIZE ----------------> A_send_query,
                             Time_to_query(3000,15000)

Time_to_query  ------------> A_nudge_query
```

In this diagram INITIALIZE and Time_to_query on the left side are the names of events. A_send_query , Time_to_query(3000,15000), and A_nudge_query are the names of actions. The rules specify that in the event of INITIALIZE, actions A_send_query and Time_to_query(3000,15000) are executed. At the event of Time_to_query, action A_nudge_query is executed. The event INITIALIZE is predefined by the EventIX tool to be the beginning of the execution of the application program, so the Event Director executes A_send_query and Time_to_query(3000,15000) at the beginning of the execution of Poller. (Actions are usually non-blocking, which means that several actions specified in a single rule take place in the order they are specified, with the actions not waiting for the completion of the previous actions.) Time_to_query is an event generated by the action Time_to_query(3000,15000) and will be explained later. A_send_query is a programmer-defined action. This action starts a process referred to as "sender". Process "sender" receives a message from the standard input and sends the message to the destination process through a specified UNIX UDP socket. The destination process of course must listen to that socket. In our design, the destination process is the IEI module. Time_to_query(3000,15000) is an action that sets a cyclic timer with initial time-out interval 3000 msec and cycle interval 15000 msec. 3000 msec after the timer is set, the timer generates Time_to_query event, and thereafter the timer generates Time_to_query event at every 15000 msec. Action A_nudge_query sends ASCII string "do_query\n" through the standard input to the program "sender".

Program ice.epa instructs its associated Event Director what to do in response to the reception of a message from IEI. Particularly, the eventdb file ice.epa instructs how to parse the incoming message and which MIB variable is to be updated. Events, actions, and rules of ice.epa are more complex than Poller, but there is a specialized software development tool for building this part of the proxy agent program, so the software developer can build the

module conveniently through the specialized graphical user interface. During the run-time, the Event Director process associated with ice.epa listens to the socket to which IEI has made a connection. Upon the message arrival, the Event Director checks the validity of the message according to ice.epa.eventdb file. If the message is valid, Event Director parses the message and maps the pieces of information to MIB variables. It then sends this information to OpenAgent so that OpenAgent can update its MIB cache accordingly.

V. DISCUSSION

This section provides evaluation of the design presented in this report. Through this evaluation, pros and cons of the particular design choices and commercial products will be implied.

A. Consolidated Management System

The advantage of connecting legacy systems (e.g. ICEbox) with the standard SNMP network management system is that it gives the central authority the ability to manage many different systems with different management protocols. The management station can have the option of directly monitoring and controlling a component of the legacy system. Also, due to the usage of the standard protocol, the central authority can have a wide variety of options in employing commercially available management software. The design presented in this report was based on two objectives: i) make the implementation of the consolidated management system as non-intrusive to existing ICEbox as possible; ii) take full advantage of the ICEbox utilities (Common Software). The design is **non-intrusive** in the sense that none of the existing ICEbox modules has to be revised or redesigned. The design only adds another ICEbox module referred to as IEI. For making ICEbox status information available to the SNMP manger, the design in this report takes the approach of polling only the EC (Element Control) module so that the proxy agent does not add much network traffic in ICEbox. In order to obtain ICEbox status information, common software utilities were heavily used in developing codes for the IEI module.

The status of the legacy system (e.g. application status, process status, mission status, system status, etc.) is not fully described by the standard Management Information Base (MIB-II). Therefore, the "iso.org.dod.internet.private " branch of the MIB object tree were used to define MIB variables used by the consolidated network management station for monitoring and controlling the legacy system (ICEbox). The proxy developer can progressively add new MIB variables to increase the scope of the central management station's ability to monitor and control the particular legacy system. For example if **new ICEbox high-level status info is requested,** a new MIB variable corresponding to that new ICEbox status information must be added to the MIB tree. Practically, the addition of such new variables into the MIB object tree can be quite easy using commercially available MIB builders such as from EventIX. Through the EventIX Graphical MIB builder, the programmer chooses the name and path in the MIB tree for a new MIB variable and adds the MIB variable to the MIB tree. Next, the IEI module must be modified so that it sends the new ICEbox status information to the proxy agent. More specifically, the C-program code of the IEI module that sends the string of text to the proxy agent must be expanded. In that string of text, the new ICEbox status information must

be added. Also, the eventdb file of the proxy agent must include the instructions that updates those new MIB variables upon the reception of the string from the IEI. These procedures are simple for programmers who are familiar with C language and trained to use the EventIX. Adding a new MIB variable and recompiling the code can take less than a few hours of a trained programmer's time. Adding several MIB variables will take more or less the same amount of time because most of the time will be spent recalling the available C-procedures and debugging the code during compilation. However, this estimate assumes a programmer who is familiar with ICEbox software. Those who do not know which COD of ICEbox describes which status information will have to consult an ICEbox developer, and this in real life can take several days.

B. SNMP Proxy Agent EventIX Tool

In the detailed design of the SNMP proxy agent for the ICEbox, communication between IEI module, Poller, and the proxy agent is designed to take place through the UNIX socket. This implies that the IEI, Poller, and the proxy agent can all reside in separate machines; hence, distributed implementation is possible.

The commercial tool EventIX was proven extremely valuable through this project of proxy development. In this project one programmer was given one week of tutorial on the use of EventIX. After the tutorial, the programmer was able to develop the entire proxy agent only with the EventIX user's manual and the technical support from the EventIX vender. The whole procedure took less than a few months. Considering that the programmer had no knowledge of the tool, this proves that the tool is easily learned. This also proves that once the programmer learns how to use EventIX, the proxy development becomes extremely speedy. The effort of developing the entire proxy agent from scratch by writing C-programming code is estimated to be years for an inexperienced programmer. The organization of the code would pose a serious challenge for a programmer who has not worked on an SNMP agent. The number of C code lines would be extremely large, and the variables and structures would be complex. EventIX hides low-level details from the programmer and allows the programmer to build a proxy agent with a conceptual understanding of the SNMP protocol only.

In implementing the design presented in this report, the proxy agent was made to run on a SUN SPARC 20 system. The consolidated network manager could send Get Request messages to this machine for two reasons; to inquire about the ICEbox status or to inquire about the status of the SPARC system itself. The SPARC system's regular SNMP agent handles the status information (MIB-II) regarding its current health and operation. The proxy agent handles only the iso.org.dod.internet.private branch of the MIB tree, where ICEbox status data are stored. The regular SNMP agent and the proxy agent are separate processes executing during the same run time, and the SNMP messages from the management station must be delivered to the proper process according the what status information is requested. In designing the proxy agent, the commercial tool, EventIX, was used, which provides an SNMP interface module described previously in section IV-C. This commercial software is designed to parse the SNMP message it receives from the management station and can be configured to forward the SNMP message according to its contents. The SNMP

interface reads the first OID (Object Identifier) of the MIB variable requested in the SNMP message and forwards the entire message to either the proxy agent or to the regular SNMP agent. If the first OID is the OID for the private MIB defined for the ICEbox status, the entire message is forwarded to the proxy agent. If the first OID is in iso.org.dod.internet.mgnt.mib-2, the entire message is forwarded to the regular SNMP agent. The commercial design of this SNMP interface could cause some problems. Consider the case that the SNMP PDU (Protocol Data Unit) contains some OIDs in .iso.org.dod.internet.mgnt.mib-2 and some in iso.org.dod.internet.private. (Some management software lists multiple OIDs in an SNMP message. For example, HP OpenView's periodical data collection asks for .iso.org.dod.internet.mgnt.mib-2.system.sysUptime of the host even if the data collection is on a private MIB variable in iso.org.dod.internet.private.) If the first OID in the PDU belongs to iso.org.dod.internet.private, the entire SNMP PDU including the request for the MIB-II variables are forwarded to the proxy agent. Because the proxy agent does not keep cache for MIB-II, the network management software does not get the response, and an error occurs. This kind of error actually happened in our use of the EventIX product (Extensible Proxy Agent Builder version 2.0). One solution is to keep a cache of the MIB-II in the private MIB. In order to implement this solution, the programmer must have a detailed knowledge of the internals of EventIX software. Another solution is to use network management software that requests only one MIB variable in each SNMP message or that requests only one type of OIDs (iso.org.dod.internet.mgnt.mib-2 branch or iso.org.dod.internet.private branch) in each SNMP PDU.

C. Use of HP OpenView for displaying ICEbox high-level status information

The graphic representation of the modular and hierarchical structure of the ICEbox system can be displayed though the OpenView graphics features. The OpenView graphics are based on a few concepts; objects, symbols, maps, and submaps. In HP OpenView Windows, an object is an internal representation of a logical or physical entity or resource that exists on the network to be displayed by the OpenView Window. [Hew93] An object might represent a physical piece of equipment, a part of a network, or even non-computer items such as a building or location. Examples of objects include a network, a computer, an interface, a software process running on a computer, building, country, region, etc. An object consists of a set of attributes, and the objects are stored internally in the OpenView Window (OVW) database. Object attributes specify the characteristics of the object such as IP address (for some types of objects), selection name, etc. (Selection name is a mandatory attribute used to identify the object.)

A symbol is a graphical representation of an object. It can be thought of as an icon that shows up in the window. Although a symbol represents an object, a symbol has its own characteristics that are apart from the object; for example, symbol status, symbol behavior. Symbols display status information of the object through the use of color; if the object has a problem such as hardware failure, the symbol appear in red color, while the color is green for normal operations. A symbol can have either explodable or executable behavior, and a user can configure these when creating a symbol. These symbol behaviors will be explained later.

The submap is a collection of related symbols that represent the actual objects on the network. A submap is a schematic representation of all or part of a network map. It is basically a window that displays symbols such as icons and lines. For example, in a submap that displays an Ethernet segment, there are icons that represent computer systems, routers, etc. and a line that represent the Ethernet bus connecting them.

A map is a set of related objects, symbols, and submaps. Users do not view a map directly; instead, users always view the submaps that make up, or reflect the map. A map is basically a hierarchical file structure containing the information regarding the graphics.

OpenView allows users to define and add new symbols to a submap. Users can define a symbol as either explodable or executable. The explodable symbol opens another submap, when clicked through the mouse, in order to describe the object further in detail. For example, by clicking an explodable symbol representing a computer system, a user in some cases can open a new submap, and that submap may have symbols representing a few network interface cards of the computer system, a software module of the computer system, etc. A process is executed when the executable symbol is clicked.

A user can utilize the explodable symbols to represent the ICEbox system hierarchically. In the submap that has the symbol for the computer hosting the proxy agent, a user can add an explodable symbol that represents the ICEbox system. A user can double click this symbol and open a new submap. In that new submap a user can add explodable symbols representing ICEbox units. In the submap opened by clicking each of these symbols, a user can again add symbols representing ICEbox modules composing that unit. In adding the symbols, users typically choose from the collection of different icons provided by OpenView. OpenView provides different types of symbols (icons) for different objects; e.g. cards, computer, connector, network, server, etc. ICEbox modules or units, however, often do not fit in such icon categories. A user can create a new icon using the HP's icon editor. However, creating icons without the icon editor of the HP workstation is nontrivial. Importing a graphic file and converting it to the OpenView icon (symbol) is possible. However, there are rather strict limitations on the size of the icon, and that can prevent users from implementing complex graphic displays. For example, suppose a user wants to make an icon that shows the picture of a certain ICEbox unit. In that case, the image file of the ICEbox unit should be converted to an OpenView icon. Due to the limitations of the size of the icons, the size of the image must be inordinately reduced even for the largest possible size of an OpenView icon. This procedure often makes the picture unintelligible. Creating an icon bigger than the maximal size is either impossible or beyond the scope of the technical support provided by the software maintenance contract. Therefore, within the limit of reasonable efforts, graphical representation of ICEbox modules or units in OpenView remains crude.

Monitoring the status of ICEbox can be done through MIB Browsing menu of the OpenView. As discussed before, the ICEbox CODs are translated into MIB variables, and the network management stations such as the HP OpenView can monitor the ICEbox system by querying the MIB variables corresponding to the CODs of interest. Under the current design of the consolidated management system, which was discussed in this report, the user of OpenView can search the MIB variables for ICEbox status using the predefined structure in the

iso.org.dod.internet.private branch of MIB tree. Therefore, the user who monitors ICEbox must have the knowledge of which ICEbox CODs (Control Objects, status information) correspond to which MIB variable in the iso.org.dod.internet.private branch.

The MIB variables for ICEbox are not standard, and general users do not have the knowledge of the mapping between COD names and MIB variables. Also, general users initially do not have knowledge of the MIB tree structure of the branch, iso.org.dod.internet.private, where the MIB variables describing the status of ICEbox are attached. Therefore, intuitive and graphic display of the ICEbox MIB variables is desired. A natural way is to associate the MIB variables describing the status of the ICEbox modules with the display of symbols representing ICEbox modules. For example, when a user selects a symbol of an ICEbox module, it is desired that the user can see what MIB variables are associated with that particular module. However, the OpenView Network Node Manager, version 3.3 does not provide facilities to associate MIB browsing and graphics that way. Therefore, OpenView Graphics do not directly guide the MIB browse.

To make the MIB browse a little more convenient for users than the built-in MIB browse, a user can define a new menu for ICEbox MIB browse through advanced customization of OpenView. A user can define menu hierarchy in line with the hierarchy of the ICEbox hierarchical graphic representation. However, even using this approach, the menu hierarchy does not have any direct association with the graphics hierarchy. A user can be guided by observing the hierarchical graphics, but cannot select or observe the corresponding MIB variables by clicking the symbol. The user has to still go through the customized menu bars to browse MIB variables. The user-defined menu will be identical for different submaps; i.e. there is no internal relation between the customized menu and graphics displays.

OpenView Network Node Manager is basically designed for the management of networks with standard protocols, and not made to have graphical representation of general systems. Even creating a line symbol that has an arrow to represent the data flow is nontrivial. The implementation of this was not covered by the HP OpenView manual, training course, or technical support under software maintenance contract during our project. The ICEbox system does not fit the general communication network paradigm. At a high-level, ICEbox can be viewed as a big data processing machine. As expected, the graphical display of the ICEbox system on HP OpenView has less capability than ICEbox's Human Computer Interface, which has been custom built for ICEbox. The graphic features of HP OpenView are not easily customized to display the data flow of the ICEbox system.

The major reason that OpenView is not as flexible as ICEbox's own GUI (graphical user interface) is that ICEbox is more than just a communication network; ICEbox has its own data flow structure. Commercial network management tools are designed to monitor and control standard communication networks. Therefore, it is only natural that the commercial tools have limitations in conveniently displaying the status of private systems. If we desire to have management software that combines the ability of all the individual management tools designed specifically for individual systems (such as the Human Computer Interface for monitoring the ICEbox), the recommended solution would involve the development of private software based on X11/motif or using a COTS GUI builder. One should note,

however, that any SNMP-based network management tool, such as OpenView, has the ability to display the status information of individual systems using private MIB variables. It is only in terms of the quality of presentation that COTS network management software is limited as compared to the capability of the custom-developed display tools. Therefore, it is reasonable to use commercial tools like HP OpenView temporarily until a private display tool is developed.

ACKNOWLEDGMENT

The authors wish to express thanks to Mr. Louis Chmura of the Naval Research Laboratory and Mr. Cary Allen for guiding through this project, and Mr. Ramji Chandramouli and Mr. Andrew Le of Bridgeway Corporation for their software and technical support.

REFERENCES

[Bri95] Bridgeway Corporation, *EventIX user's manual,* Release 2.1. 1995.

[Com91] D.E. Comer, *Internetworking with TCP/IP*, volume 1, Prentice-Hall, Englewood Cliffs, NJ, second edition, 1991.

[Fei95] Sidnie Feit, SNMP: a guide to network management, McGraw Hill, New York, 1995.

[Hew93] Hewlett Packard, *HP OpenView Windows User's Guide*, Manual Part Number: J2311-90002, 1993.

[Ros94] M.T. Rose, *The simple book: an introduction to internet management.* PTR Prentice-Hall, Englewood Cliffs, NJ, second edition, 1994.